Introduction to AI Assignment 3 Report

Onur YILMAZ S009604

Abstract

This assignment's aim is to classify variation of the MNIST dataset of handwritten digits. I adopt feature selection and construction techniques together with three main machine learning algorithms: Naive Bayes, K-Nearest Neighbor and Decision Tree algorithm. Report provides analyzing and assessing the parameter selection process and the performance of each algorithm. Report is concluded with the result and discussion of each algorithm.

The MNIST Data

The MNIST character dataset is an image classification problem. An important aspect of the data is that the images have been centred and scaled to the characters, reduced to black and white images and the resolution set uniform. Classification algorithms applied to this dataset are then left with higher-level logic, finding significant edges and pixels to distinguish between characters.

The whole dataset contains 70,000 images and every Image has 784 features because every image is 28x28 pixels big. Each image represents the intensity of one pixel from 0 (white) to 255 (black). The dataset is split into a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centred in a fixed-size image.

Let's look at one image:

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

digit = X_label[66043]
digit_image = digit.reshape(28, 28)
plt.imshow(digit_image, cmap = matplotlib.cm.binary,
           interpolation="nearest")
plt.axis("off")
plt.show()
```

The MNIST dataset downloaded from the original website called ‘ yann.lecun.com’ and the file was loaded into Python3 via read_image_file and read_label_file methods(you can see below implementation details).

Mnist Data Files

MNIST database is provided for you in the folder mnist data/, which consists of four files:

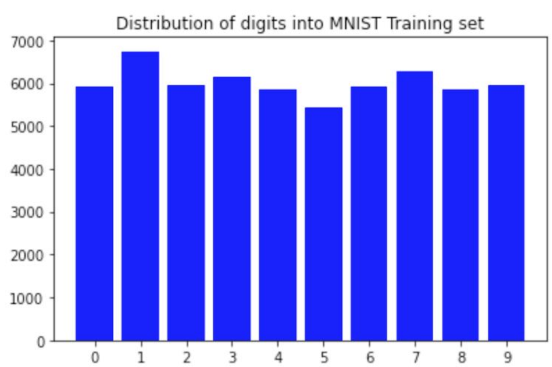
1. train-images-idx3-ubyte, which contains 60,000 28 × 28 grayscale training im- ages, each representing a single handwritten digit.
2. train-labels-idx1-ubyte,whichcontainstheassociated60,000labelsforthe train- ing images.
3. t10k-images-idx3-ubyte, which contains 10,000 28 × 28 grayscale test images, each representing a single handwritten digit.
4. t10k-labels-idx1-ubyte, which contains the associated 10,000 labels for the test images.

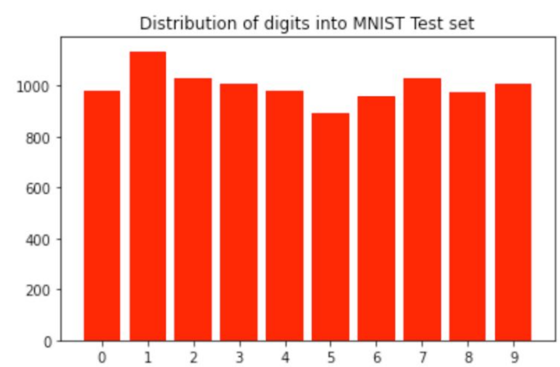
I read and uploaded the mnist dataset shown on figure[0].

```python
def read_image_file(filename, images):
    width = 28
    height = 28
    N = images

    f = gzip.open(filename, 'r')
    f.read(16) # skip preamble, 16 bytes
    buffer = f.read(width * height * N) # read in "N" images as binary data
    data = np.frombuffer(buffer, dtype='uint8') # convert binary data to integers : 0 - 255
    data = data.reshape(N, width, height, 1) # reshape to Nx28x28x1 (only 1 color channel, b/w)
    f.close()

    return data

def read_label_file(filename, labels):
    N = labels

    f = gzip.open(filename, 'r')
    f.read(8) # skip preamble, 8 bytes
    buffer = f.read(N) # read in "N" labels as binary data
    data = np.frombuffer(buffer, dtype='uint8') # convert binary data to integers : 0 - 255
    f.close()

    return data
```

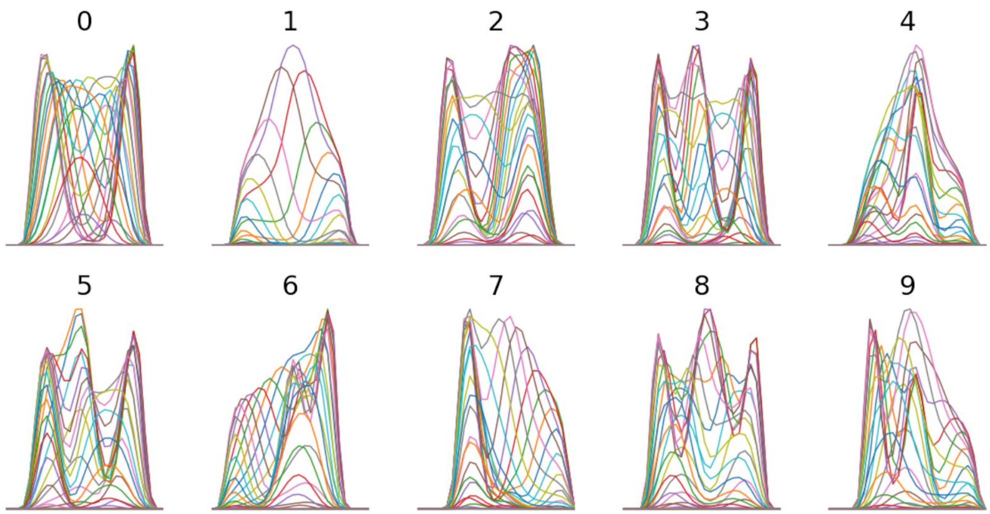
Figure[0]

MNIST Training and Test Dataset Distribution





The Mean Values of Digits



K-Nearest Neighbor (KNN) Classifier

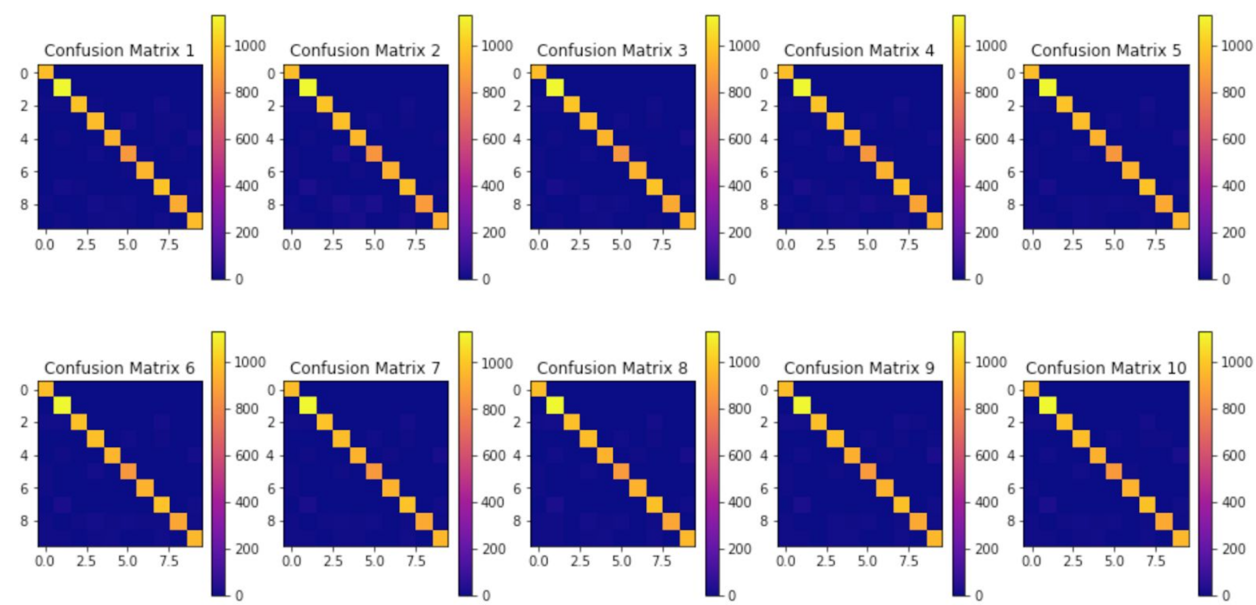
The K Nearest Neighbor algorithm works by looping through each value in the test data, determines the nearest neighbour, then predicts the result based on the point that it's closest to. From the input images on the projected space, a k-nearest neighbour classification model, which classifies images into one of the ten k-value classes [0,1,2,3,4,5,6,7,8,9] using the euclidean distance metric. Experiment is made with changing different parameters to show overall knn performance.

Experiments:

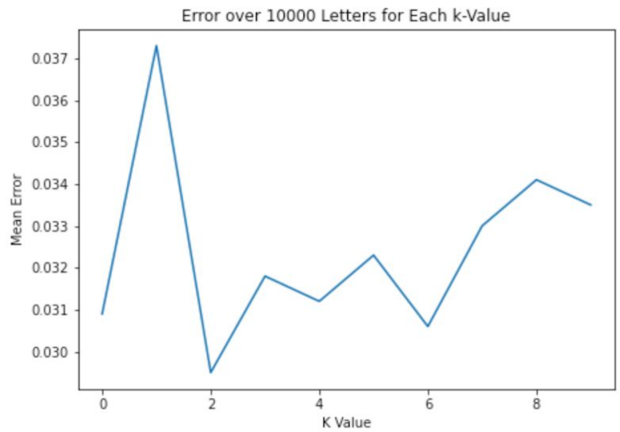
a) Iteration K-value with constant train and test size.

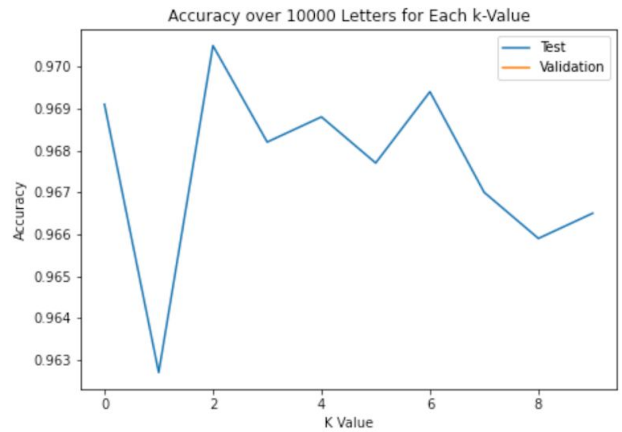
Figure[1]

K-value	Accuracy	Time	Train & Test Size
1	0.9691	387.26s	60000 & 10000
2	0.9627	424.47s	60000 & 10000
3	0.9705	448.55s	60000 & 10000
4	0.9682	461.11s	60000 & 10000
5	0.9688	483.53s	60000 & 10000
6	0.9677	425.42s	60000 & 10000
7	0.9694	395.15s	60000 & 10000
8	0.967	441.97s	60000 & 10000
9	0.9659	448.64s	60000 & 10000
10	0.9665	442.41s	60000 & 10000



```
Highest test accuracy: 0.9705 with k-value of 3
```

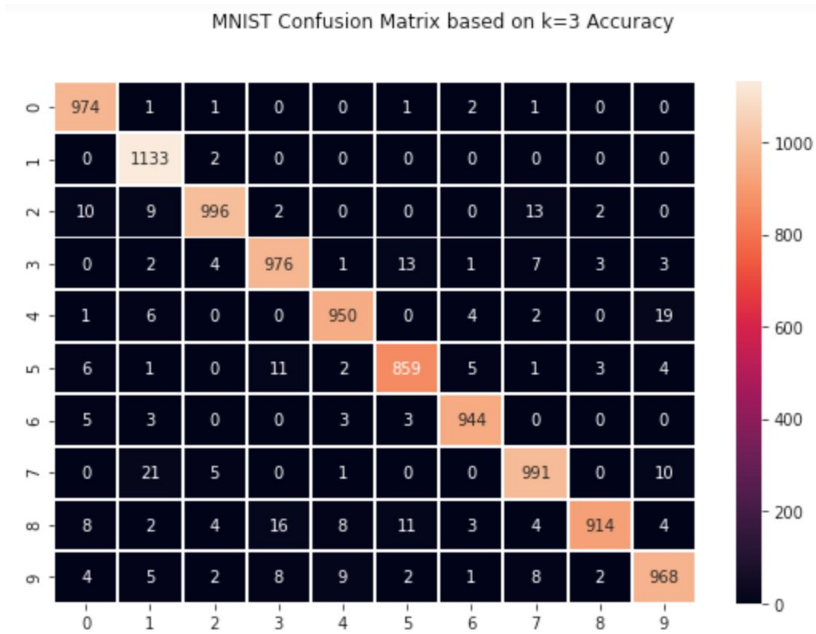




Although the results are closer to each other, high accuracy is achieved when k=3 with %97.05 accuracy rate, irrespective of the training and test data size. By keeping these parameters as a constant, the model was trained on 60000 training images and tested on 10000 images.

Detailed Result of K-Val = 3

	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.96	1.00	0.98	1135
2	0.98	0.97	0.97	1032
3	0.96	0.97	0.96	1010
4	0.98	0.97	0.97	982
5	0.97	0.96	0.96	892
6	0.98	0.99	0.98	958
7	0.96	0.96	0.96	1028
8	0.99	0.94	0.96	974
9	0.96	0.96	0.96	1009
accuracy			0.97	10000
macro avg	0.97	0.97	0.97	10000
weighted avg	0.97	0.97	0.97	10000



	Accuracy
Digit 0	99.387755
Digit 1	99.823789
Digit 2	96.511628
Digit 3	96.633663
Digit 4	96.741344
Digit 5	96.300448
Digit 6	98.538622
Digit 7	96.400778
Digit 8	93.839836
Digit 9	95.936571

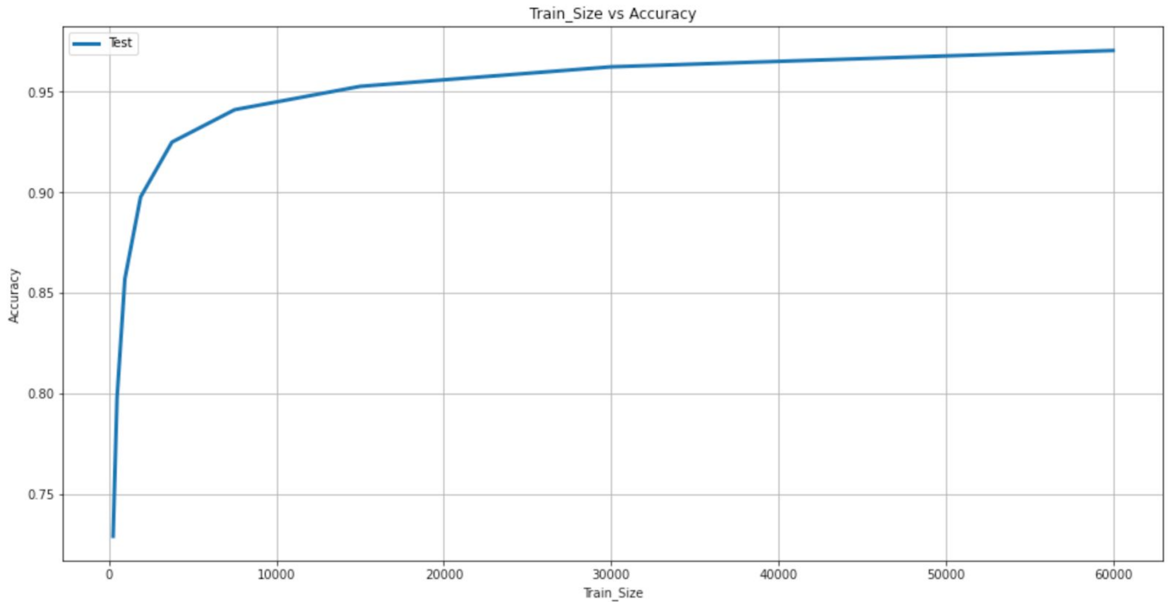
```
Accuracy of Classifier on Test Images:  0.9705
```

b) Changing Training Size with Constant K-Value and Test Size

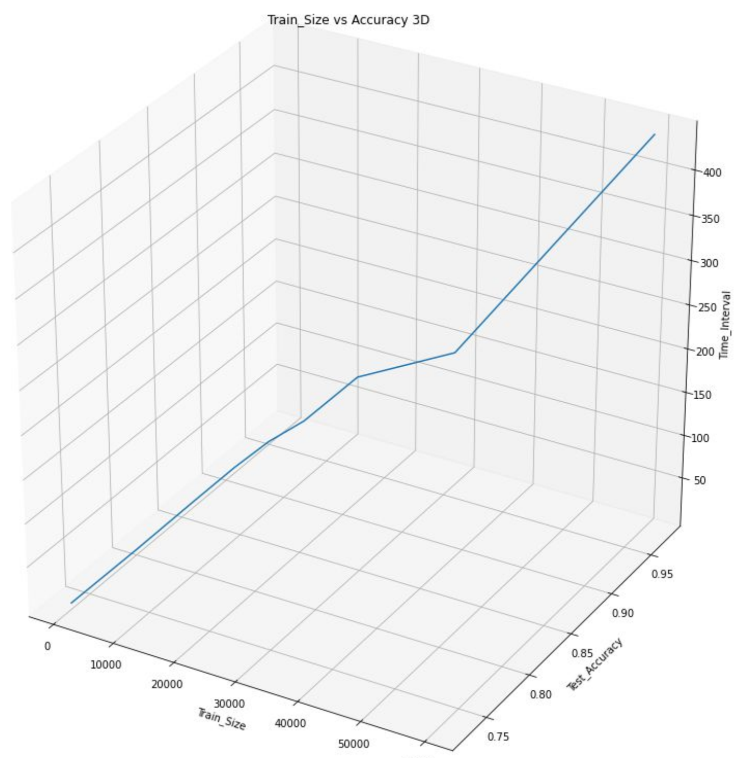
Figure[2]

Train Size	Accuracy	Time	K-Val & Test Size
60000	0.9705	444.21s	k=3 & 10000
30000	0.9624	150.76s	k=3 & 10000
15000	0.9527	101.03s	k=3 & 10000
7500	0.9411	43.83s	k=3 & 10000
3750	0.925	25.36s	k=3 & 10000
1875	0.8977	14.08s	k=3 & 10000
937	0.8571	6.98s	k=3 & 10000
468	0.798	2.82s	k=3 & 10000
234	0.7293	2.04s	k=3 & 10000
117	0.7015	1.78s	k=3 & 10000

- **Train Size and Accuracy Correlation Graph**



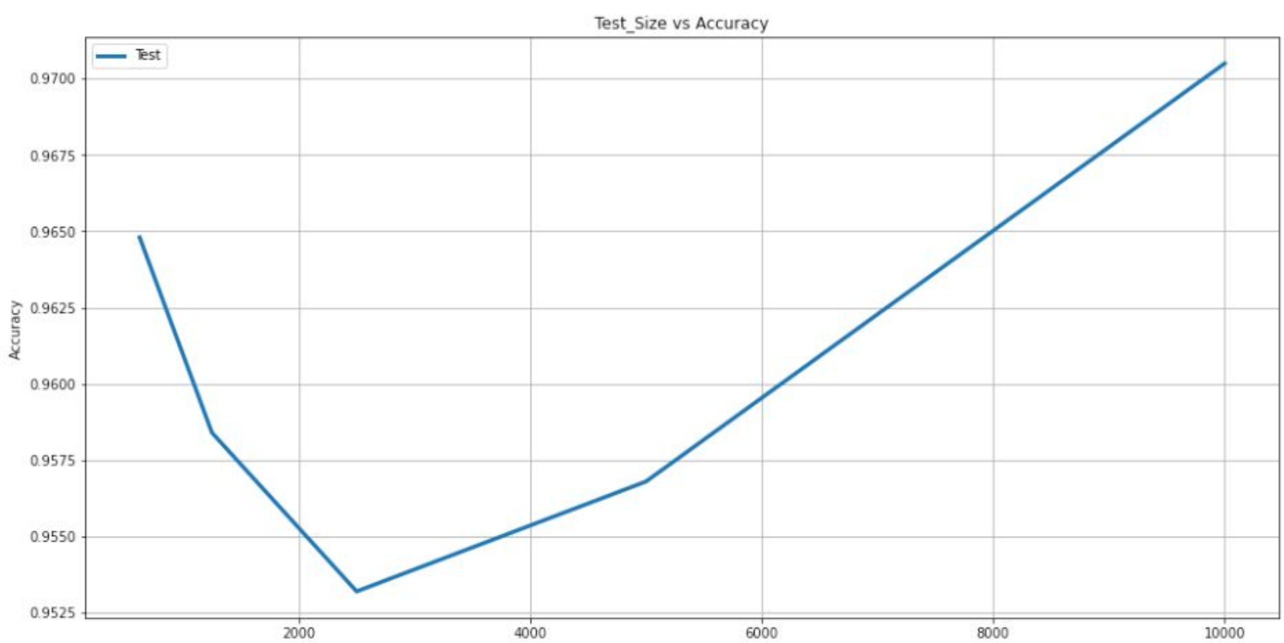
- **Train Size and Accuracy Correlation with Time factor**



c) Changing Testing Size with Constant K-Value and Train Size.

Figure[3]

Test Size	Accuracy	Time	K-Val & Train Size
10000	0.9705	487.60s	k=3 & 60000
5000	0.9568	258.61s	k=3 & 60000
2500	0.9532	144.94s	k=3 & 60000
1250	0.9584	101.84s	k=3 & 60000
675	0.9648	75.41s	k=3 & 60000
337	0.9754	54.21s	k=3 & 60000



Result and Discussion:

I executed a k-nearest neighbor(k-nn) classification algorithm for classifying the MNIST digit images in the test database using the feature vector of the training database.

The algorithm is executed with the value of k is between 1-10. and the graphical representation of the accuracy of classification in using various k values are shown in figure 1. The overall classification results are listed out in the table and figure1 table clearly evident that the optimal value of k is 3 for classification of MNIST numerical digits by using k-nearest neighbor classification technique.

Furthermore, the recognition rate of the individual digits shown in the table and overall recognition rate of the test database is 97.05%. Besides, the Knn algorithm is executed with different train size and test size parameters, hence, figure[2] clearly shows how train size and test size affects the overall accuracy result. Finally, when we compare with other classification algorithms, knn provides the best result among them.

Naive Bayes Classifiers

Naive Bayes is a classifier which uses Bayes Theorem. It calculates the probability for membership of a data-point to each class and assigns the label of the class with the highest probability. Naive Bayes is one of the fastest and simple classification algorithms and is usually used as a baseline for classification problems.

1) Gaussian Naive Bayes Classifier

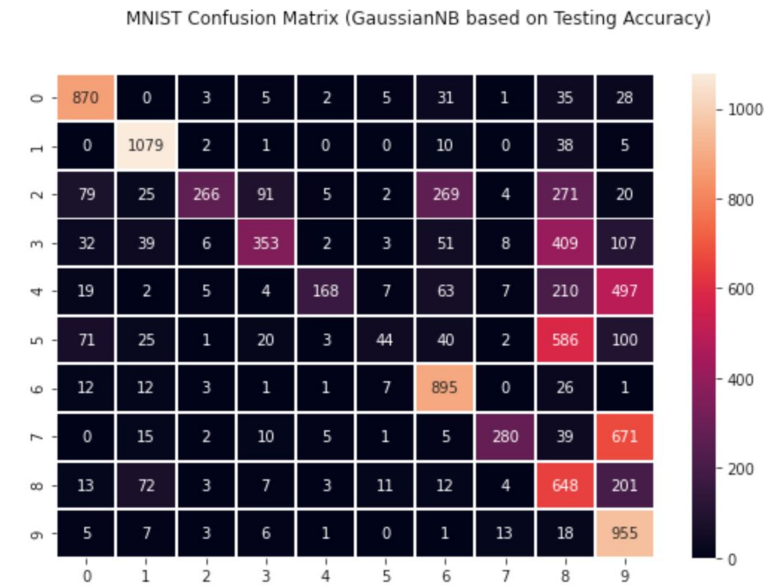
GaussianNB implements the Gaussian Naive Bayes algorithm for classification. Algorithm uses below formula to classify dataset.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Experiments:

a) Gaussian NB performance with original dataset size

```
Train time elapsed: 15.88s
Test time elapsed: 1.54s
Training accuracy: 56.49%
Testing accuracy: 55.58%
                  === Classification Report ===
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       980
           1       0.85      0.95      0.90      1135
           2       0.90      0.26      0.40      1032
           3       0.71      0.35      0.47      1010
           4       0.88      0.17      0.29       982
           5       0.55      0.05      0.09       892
           6       0.65      0.93      0.77       958
           7       0.88      0.27      0.42      1028
           8       0.28      0.67      0.40       974
           9       0.37      0.95      0.53      1009

    accuracy                           0.56     10000
   macro avg       0.69      0.55      0.51     10000
weighted avg       0.69      0.56      0.52     10000


Accuracy of Classifier on Test Images:  0.5558
```



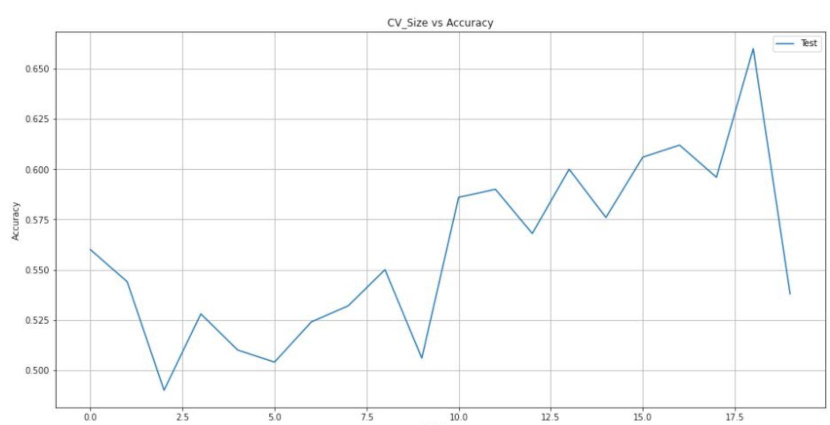
	Accuracy
Digit 0	88.775510
Digit 1	95.066079
Digit 2	25.775194
Digit 3	34.950495
Digit 4	17.107943
Digit 5	4.932735
Digit 6	93.423800
Digit 7	27.237354
Digit 8	66.529774
Digit 9	94.648167

Figure[4]

b) GaussianNB Cross Validation

The choice of the cross-validation value can impact the performance of the classifier. Hence, I conduct a cross-validation procedure to evaluate the obtained accuracies for Gaussian NB in the range [0,20].

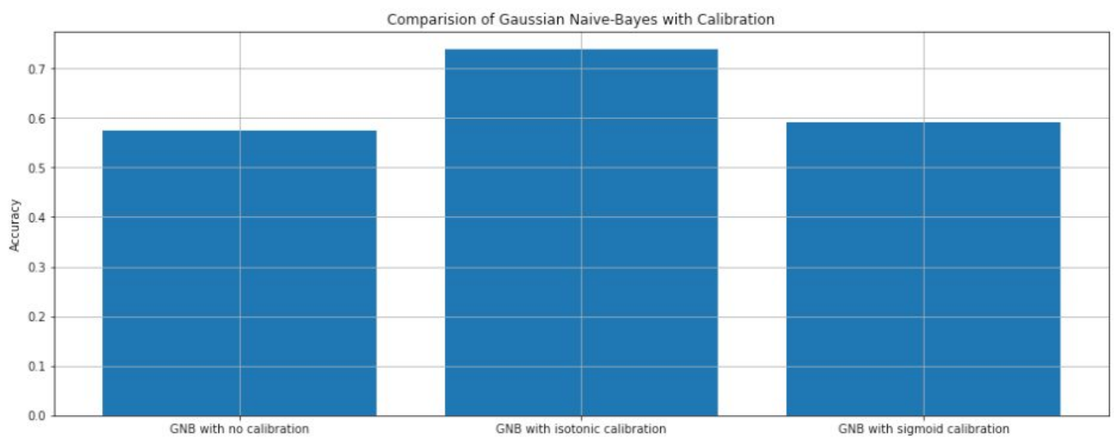
Cross Validation of GaussianNB




```
array([0.56 , 0.544, 0.49 , 0.528, 0.51 , 0.504, 0.524, 0.532, 0.55 ,
       0.506, 0.586, 0.59 , 0.568, 0.6  , 0.576, 0.606, 0.612, 0.596,
       0.66 , 0.538])
```

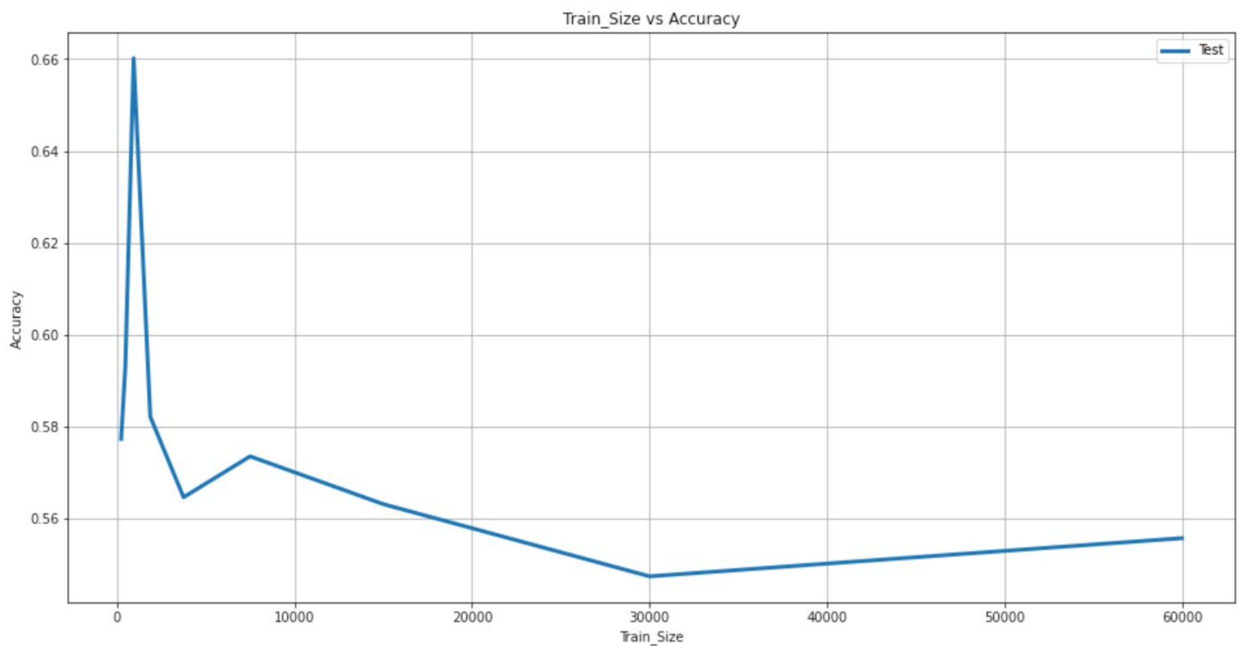
c) Gaussian Naive-Bayes Optimization with Calibration Methods

Not all classifiers provide well-calibrated probabilities, some being over-confident while others being under-confident. Thus, I compared different probability estimators' performance.



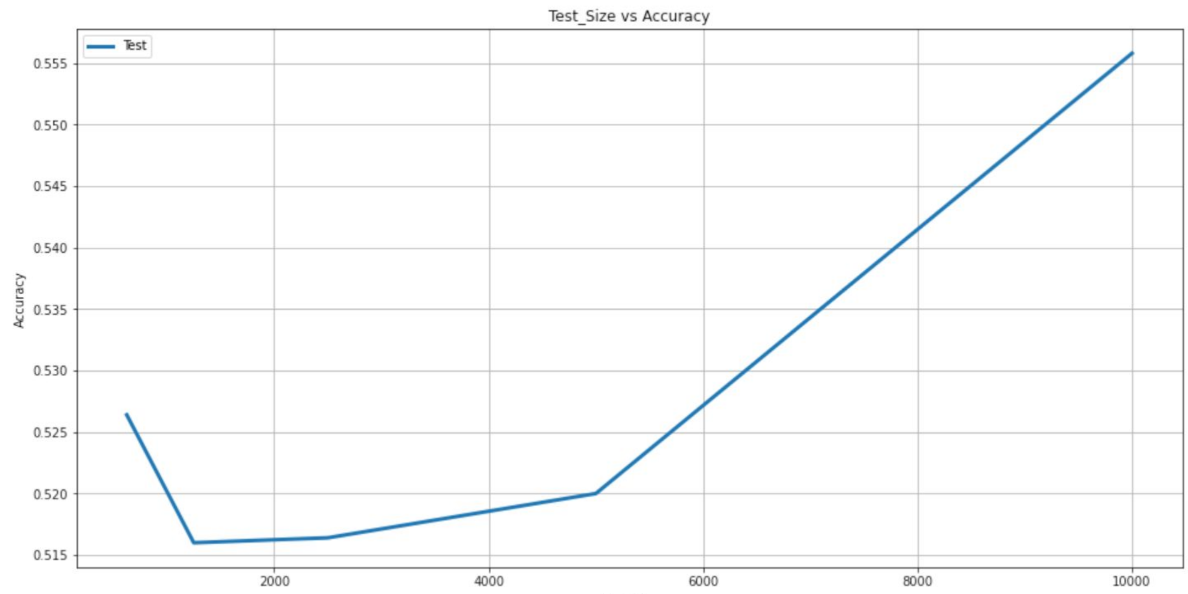
d) Changing training size with constant test size

Train Size	Accuracy	Time	Test Size
60000	0.5558	1.68s	10000
30000	0.5475	1.20s	10000
15000	0.5632	0.93s	10000
7500	0.5736	0.82s	10000
3750	0.5647	0.74s	10000
1875	0.5822	0.70s	10000
937	0.6602	0.69s	10000
468	0.5931	0.69s	10000
234	0.5774	0.68s	10000
117	0.5432	0.62s	10000



e) Changing testing size with constant train size.

Test Size	Accuracy	Time	Train Size
10000	0.5558	1.71s	60000
5000	0.52	1.26s	60000
2500	0.5164	1.14s	60000
1250	0.516	1.03s	60000
675	0.5264	1.01s	60000
337	0.5036	0.98s	60000



Concluding Remarks

The Gaussian Naive Bayes classifier provides the lowest accuracy result among the other naive Bayes. Looking at the confusion matrix figure 4, we can observe that (5,8), (5,9), (4,8), (4,9), (7,9) are some of the combinations where the classifier is confused in predicting the right label. However, when I utilized the calibration method, GaussianNB results advanced %12 more. Moreover, train and test size has positive correlation for accuracy percentage, 'd' and 'e' sections statistically demonstrate this correlation.

2) Multinomial Naive Bayes

Multinomial Naive Bayes estimates the conditional probability of a particular word/term/token given a class as the relative frequency of term t in documents belonging to class c:

$$P\ (t \mid c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

Thus, this variation takes into account the number of occurrences of term t in training documents from class c, including multiple occurrences.[11]

Experiments:

a) Multinomial NB performance with original dataset size

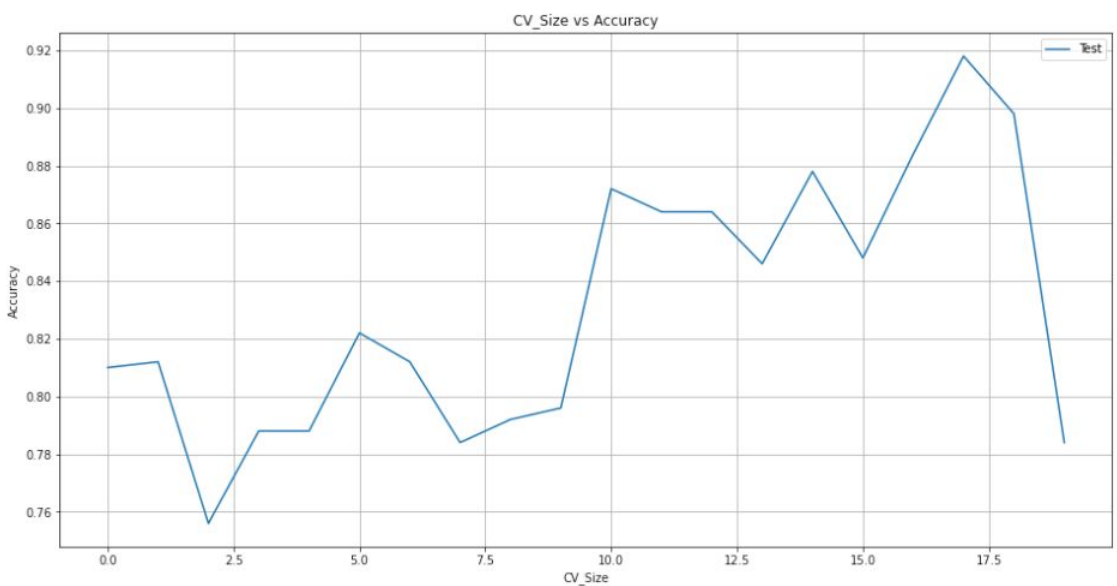
```
Train time elapsed: 0.82s
Test time elapsed: 0.11s
Training accuracy: 82.53%
Testing accuracy: 83.65%
                  === Classification Report ===
              precision    recall  f1-score   support

           0       0.92      0.93      0.93       980
           1       0.91      0.93      0.92      1135
           2       0.90      0.83      0.86      1032
           3       0.80      0.84      0.82      1010
           4       0.84      0.75      0.79       982
           5       0.86      0.66      0.75       892
           6       0.89      0.90      0.89       958
           7       0.94      0.84      0.88      1028
           8       0.66      0.80      0.72       974
           9       0.71      0.86      0.78      1009

    accuracy                           0.84     10000
   macro avg       0.84      0.83      0.84     10000
weighted avg       0.84      0.84      0.84     10000

Accuracy of Classifier on Test Images:  0.8365
```

MNIST Confusion Matrix (MultinomialNB based on Testing Accuracy)

	Accuracy
Digit 0	93.061224
Digit 1	93.480176
Digit 2	83.139535
Digit 3	84.257426
Digit 4	74.541752
Digit 5	66.143498
Digit 6	89.770355
Digit 7	83.754864
Digit 8	79.774127
Digit 9	85.530228

b) MultinomialNB Cross Validation

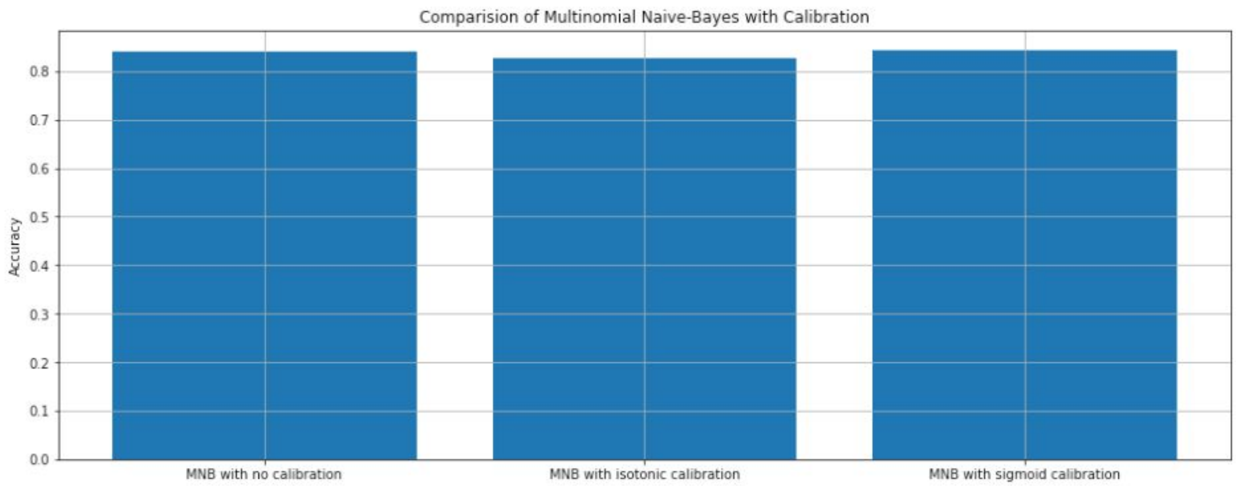
```
Cross Validation of MultinomialNB
```




```
Max Accuracy: 0.918

array([0.81 , 0.812, 0.756, 0.788, 0.788, 0.822, 0.812, 0.784, 0.792,
       0.796, 0.872, 0.864, 0.864, 0.846, 0.878, 0.848, 0.884, 0.918,
       0.898, 0.784])
```

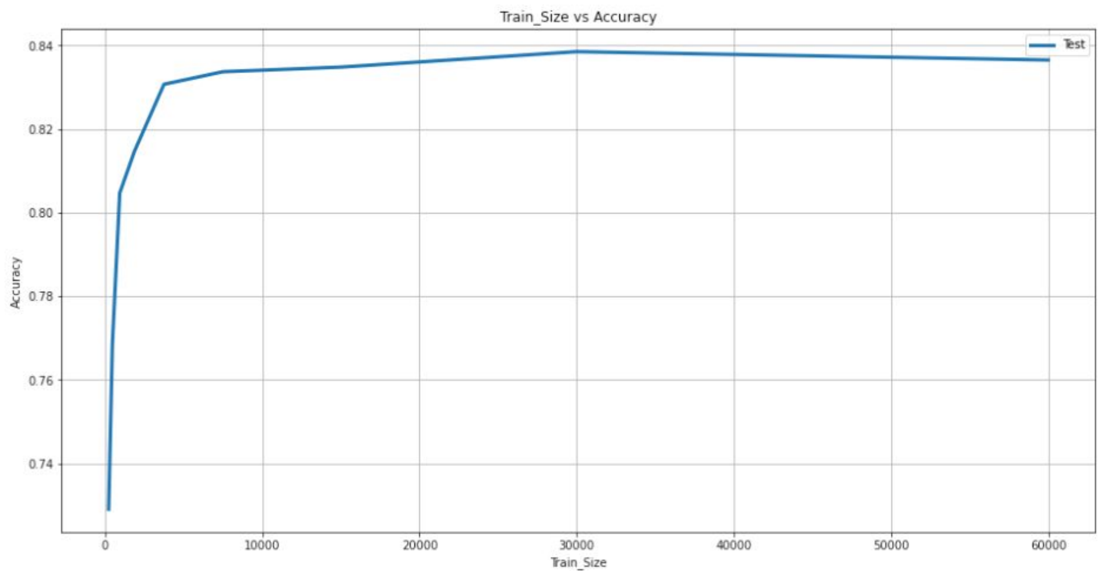
c) Multinomial Naive-Bayes Optimization with Calibration Methods

```
0.8399
0.8277
0.8426
```



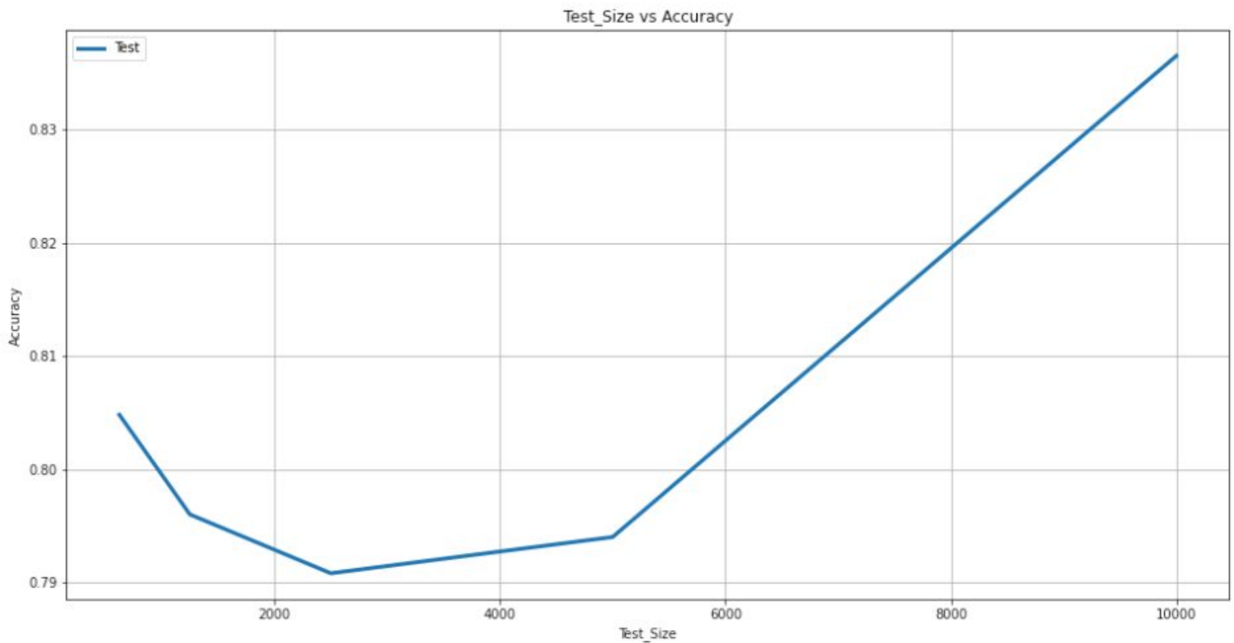
d) Changing training size with constant test size

Train Size	Accuracy	Time	Test Size
60000	0.8365	13.54s	10000
30000	0.8385	4.55s	10000
15000	0.8348	2.39s	10000
7500	0.8337	1.06s	10000
3750	0.8307	0.57s	10000
1875	0.8147	0.34s	10000
937	0.8047	0.23s	10000
468	0.7685	0.15s	10000
234	0.7291	0.13s	10000
117	0.7145	0.12s	10000



e) Changing testing size with constant train size.

Test Size	Accuracy	Time	Train Size
10000	0.8365	11.90s	60000
5000	0.794	11.54s	60000
2500	0.7908	13.25s	60000
1250	0.796	13.77s	60000
675	0.8048	13.08s	60000
337	0.8438	12.54s	60000



Conclusion and Discussion

Experiment shows that Multinomial Naive Bayes performs significantly better than Gaussian Naive Bayes on MNIST dataset. Multinomial Naive Bayes reaches an accuracy of about %83.65 percent only which is quite efficient. Furthermore, the digit accuracy table shows that MNB has better performance on 0, 1 and 6 digits and it has slightly less performance on 5, 4 and 8 digits. Moreover, 'e' experiment which is changing train size with constant test size clearly evidence that it is very difficult to achieve high accuracies with less amount of data and more data will lead to greater accuracies. Here, I can conclude that Multinomial Naïve Bayes provides greater accuracy and it also has second best accuracy among Naive Bayes classifiers which is shown on the comparison table.

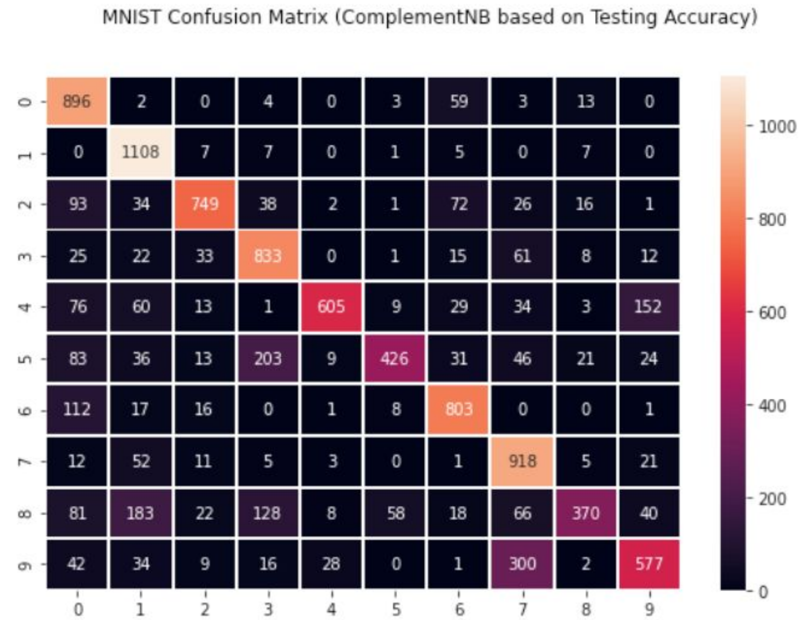
3) Complement Naive Bayes Classifier

CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the *complement of* each class to compute the model's weights.

Experiments:

a) Complement NB performance with original dataset size

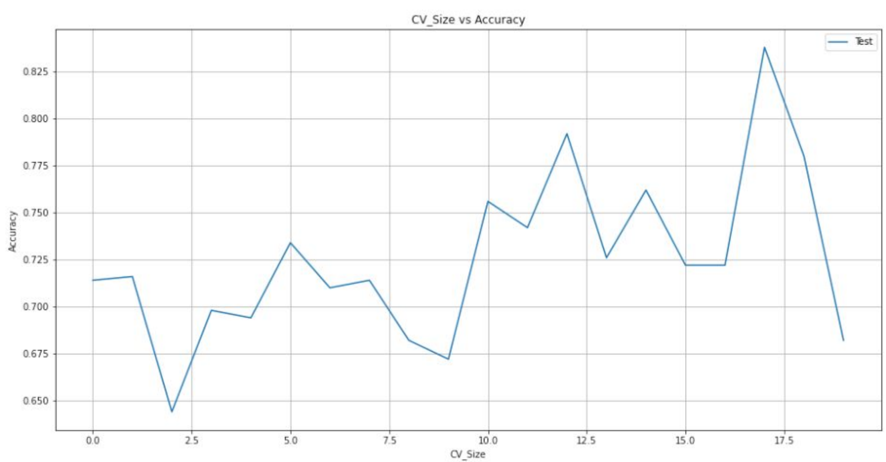
```
Train time elapsed: 0.69s
Test time elapsed: 0.09s
Training accuracy: 71.54%
Testing accuracy: 72.85%
                  === Classification Report ===
              precision    recall  f1-score   support

           0       0.63      0.91      0.75       980
           1       0.72      0.98      0.83      1135
           2       0.86      0.73      0.79      1032
           3       0.67      0.82      0.74      1010
           4       0.92      0.62      0.74       982
           5       0.84      0.48      0.61       892
           6       0.78      0.84      0.81       958
           7       0.63      0.89      0.74      1028
           8       0.83      0.38      0.52       974
           9       0.70      0.57      0.63      1009

    accuracy                           0.73     10000
   macro avg       0.76      0.72      0.71     10000
weighted avg       0.76      0.73      0.72     10000

Accuracy of Classifier on Test Images:  0.7285
```



	Accuracy
Digit 0	91.428571
Digit 1	97.621145
Digit 2	72.577519
Digit 3	82.475248
Digit 4	61.608961
Digit 5	47.757848
Digit 6	83.820459
Digit 7	89.299611
Digit 8	37.987680
Digit 9	57.185332

b) Complement NB Cross Validation

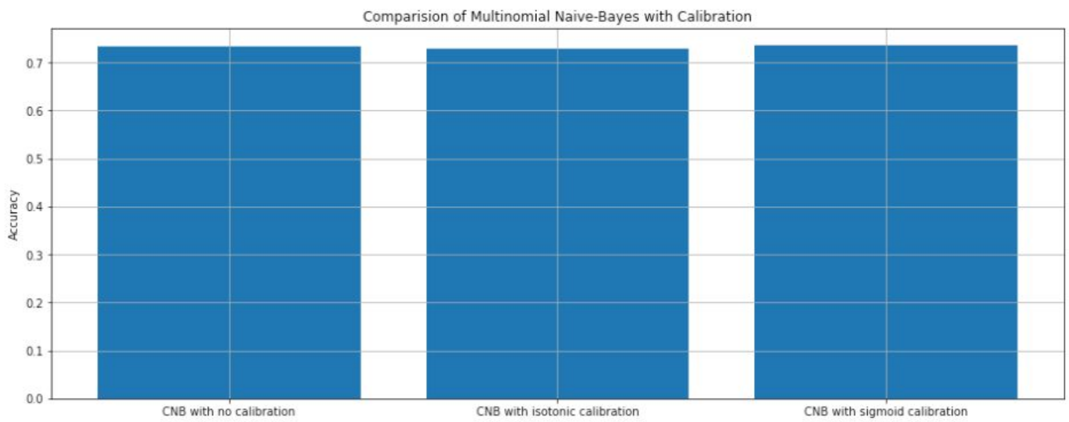
Cross Validation of ComplementNB




```
Max Accuracy: 0.838

array([0.714, 0.716, 0.644, 0.698, 0.694, 0.734, 0.71 , 0.714, 0.682,
       0.672, 0.756, 0.742, 0.792, 0.726, 0.762, 0.722, 0.722, 0.838,
       0.78 , 0.682])
```

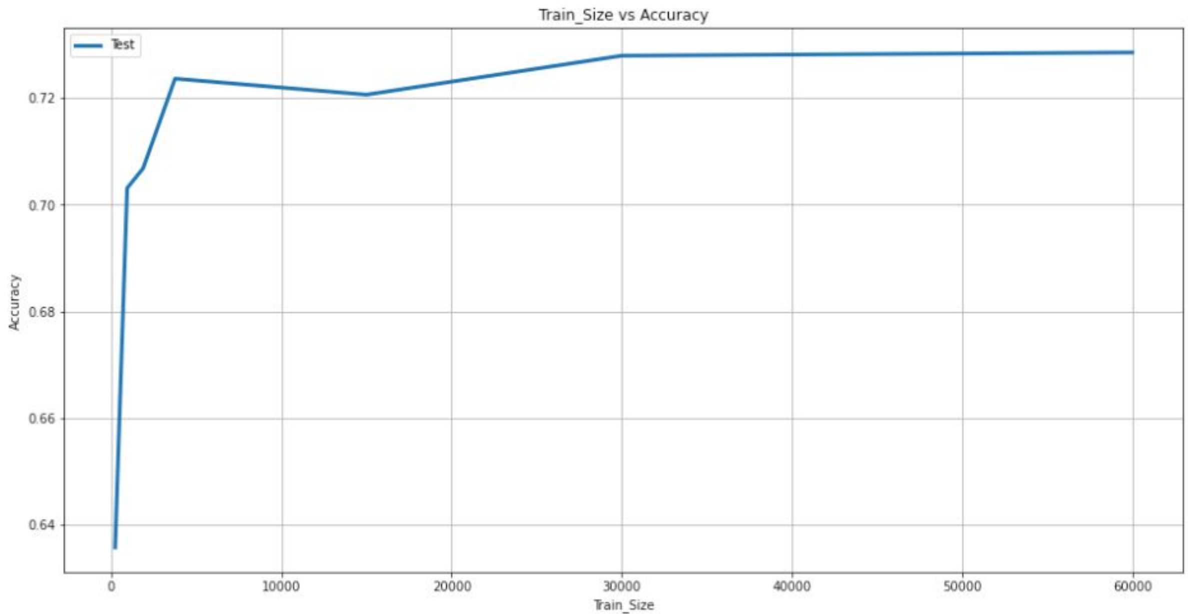
c) Complement Bayes Optimization with calibration Methods

```
0.7331
0.7300333333333333
0.7356
```



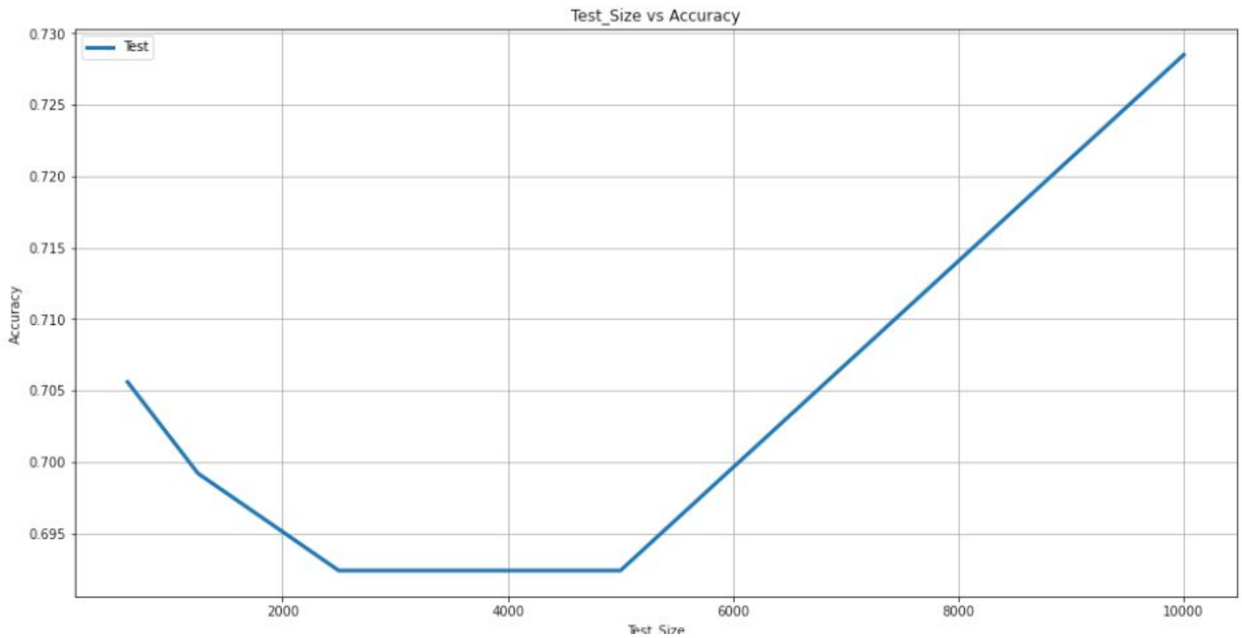
d) CNB Changing training size with constant test size

Train Size	Accuracy	Time	Test Size
60000	0.7285	14.81s	10000
30000	0.7279	6.34s	10000
15000	0.7206	3.23s	10000
7500	0.7226	1.09s	10000
3750	0.7236	0.57s	10000
1875	0.7068	0.31s	10000
937	0.6585	0.35s	10000
468	0.6358	0.20s	10000
234	0.6258	0.12s	10000
117	0.7145	0.12s	10000



e) CNB Changing testing size with constant train size.

Test Size	Accuracy	Time	Train Size
10000	0.7285	16.62s	60000
5000	0.6924	16.32s	60000
2500	0.6924	16.82s	60000
1250	0.6992	16.30s	60000
675	0.7056	16.53s	60000
337	0.7046	16.21s	60000



Conclusion and Discussion

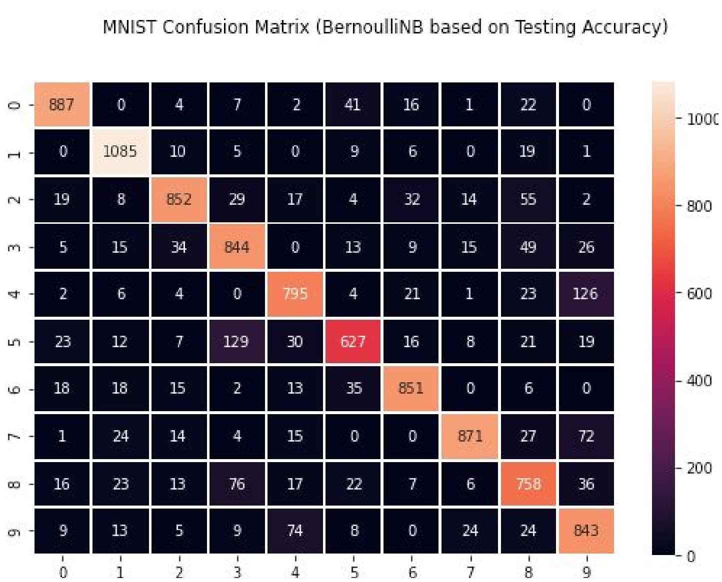
Complement Naive Bayes classifier presents an average accuracy result within the other Naive Bayes classifiers. Digit accuracy table displays that CNB has high achievement on 1, 0, 7, and digits however, 8, 5, 9 and digits has the lowest accuracy percentage. Overall accuracy score equals %72.85. Moreover, the accuracy rate raised %83.25 with improving cross-validation value.

3) Bernoulli Naive Bayes Classifier

The Bernoulli variation generates a Boolean indicator about each term of the vocabulary equal to 1 if the term belongs to the examining document and 0 if it does not. [12]

a) Bernoulli NB performance with original dataset size

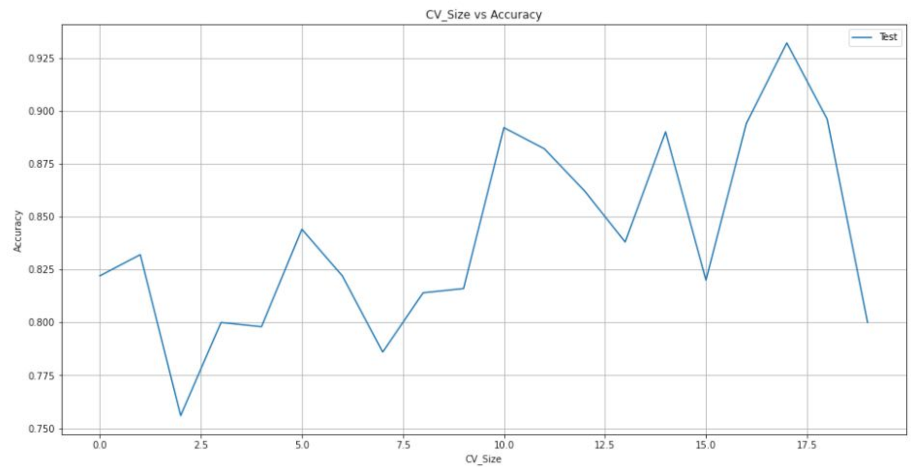
```
Train time elapsed: 1.38s
Test time elapsed: 0.16s
Training accuracy: 83.12%
Testing accuracy: 84.13%
                  === Classification Report ===
              precision    recall  f1-score   support

           0       0.91      0.91      0.91       980
           1       0.90      0.96      0.93      1135
           2       0.89      0.83      0.86      1032
           3       0.76      0.84      0.80      1010
           4       0.83      0.81      0.82       982
           5       0.82      0.70      0.76       892
           6       0.89      0.89      0.89       958
           7       0.93      0.85      0.89      1028
           8       0.75      0.78      0.77       974
           9       0.75      0.84      0.79      1009

    accuracy                           0.84     10000
   macro avg       0.84      0.84      0.84     10000
weighted avg       0.84      0.84      0.84     10000

Accuracy of Classifier on Test Images:  0.8413
```



	Accuracy
Digit 0	90.510204
Digit 1	95.594714
Digit 2	82.558140
Digit 3	83.564356
Digit 4	80.957230
Digit 5	70.291480
Digit 6	88.830898
Digit 7	84.727626
Digit 8	77.823409
Digit 9	83.548067

b) Bernoulli NB Cross Validation

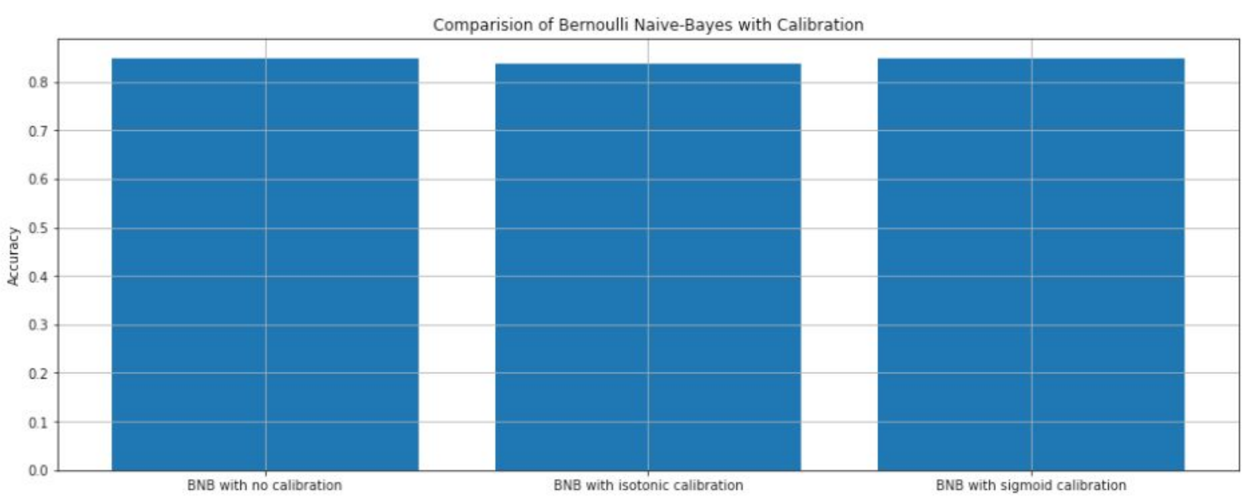
```
Cross Validation of BernoulliNB
```




```
Max Accuracy: 0.932

array([0.822, 0.832, 0.756, 0.8  , 0.798, 0.844, 0.822, 0.786, 0.814,
       0.816, 0.892, 0.882, 0.862, 0.838, 0.89 , 0.82 , 0.894, 0.932,
       0.896, 0.8  ])
```

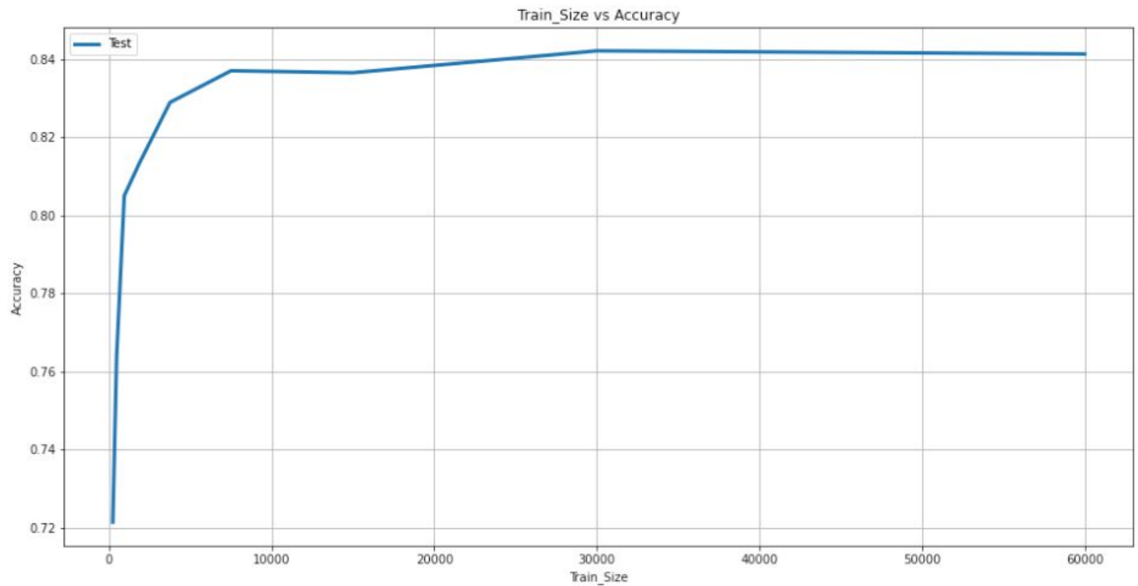
c) Bernoili Naive Bayes Optimization with calibration Methods

```
0.8472
0.83575
0.8474
```



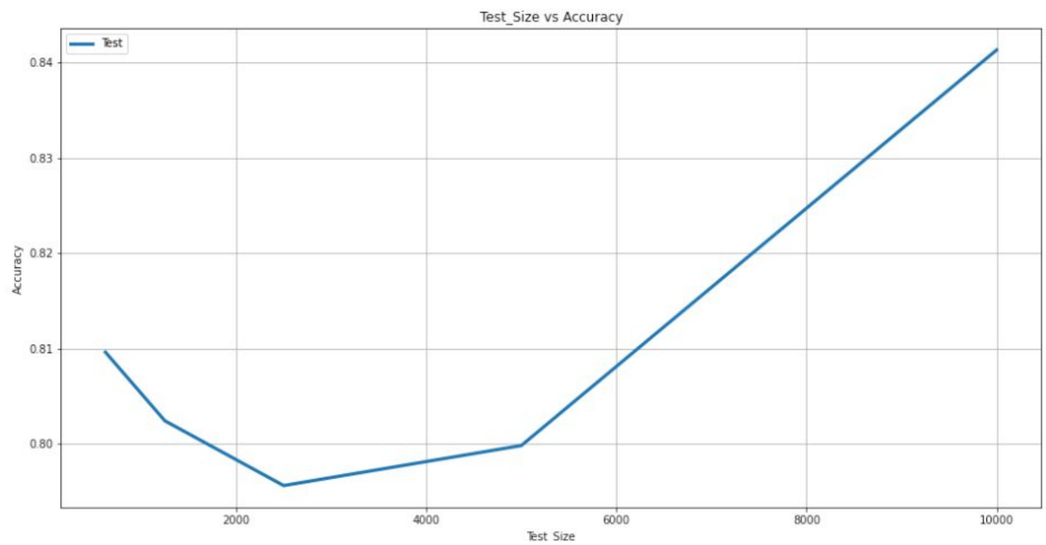
d) BNB Changing training size with constant test size

Train Size	Accuracy	Time	Test Size
60000	0.8413	16.70s	10000
30000	0.8421	8.06s	10000
15000	0.8348	3.15s	10000
7500	0.837	1.36s	10000
3750	0.8289	0.77s	10000
1875	0.8134	0.31s	10000
937	0.805	0.35s	10000
468	0.7685	0.15s	10000
234	0.7291	0.12s	10000
117	0.7145	0.12s	10000



e) Changing testing size with constant train size.

Test Size	Accuracy	Time	Train Size
10000	0.8413	15.63s	60000
5000	0.7998	13.26s	60000
2500	0.7956	12.86s	60000
1250	0.8024	13.08s	60000
675	0.8096	12.83s	60000
337	0.8046	11.83s	60000



Conclusion and Discussion

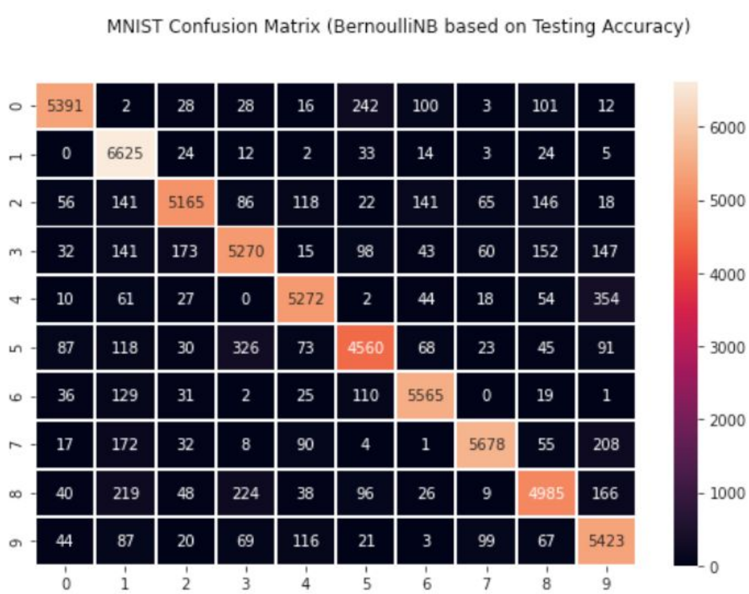
Bernoulli Naive Bayes classifier presents a satisfactory digit accuracy percentage. Although overall accuracy rate equals to %84.13, minimum digit accuracy percentage approaches %70 and maximum rate matches %95. Further, the accuracy rate grew %93.20 by iterating cross-validation values. Ultimately, the overall accuracy scale can be augmented by growing train data size and appending calibration methods.

4) Categorical Bayes Classifier

The categorical Naive Bayes classifier is suitable for classification with discrete features that are categorically distributed. The categories of each feature are drawn from a categorical distribution.

a) Categorical NB performance with original dataset size

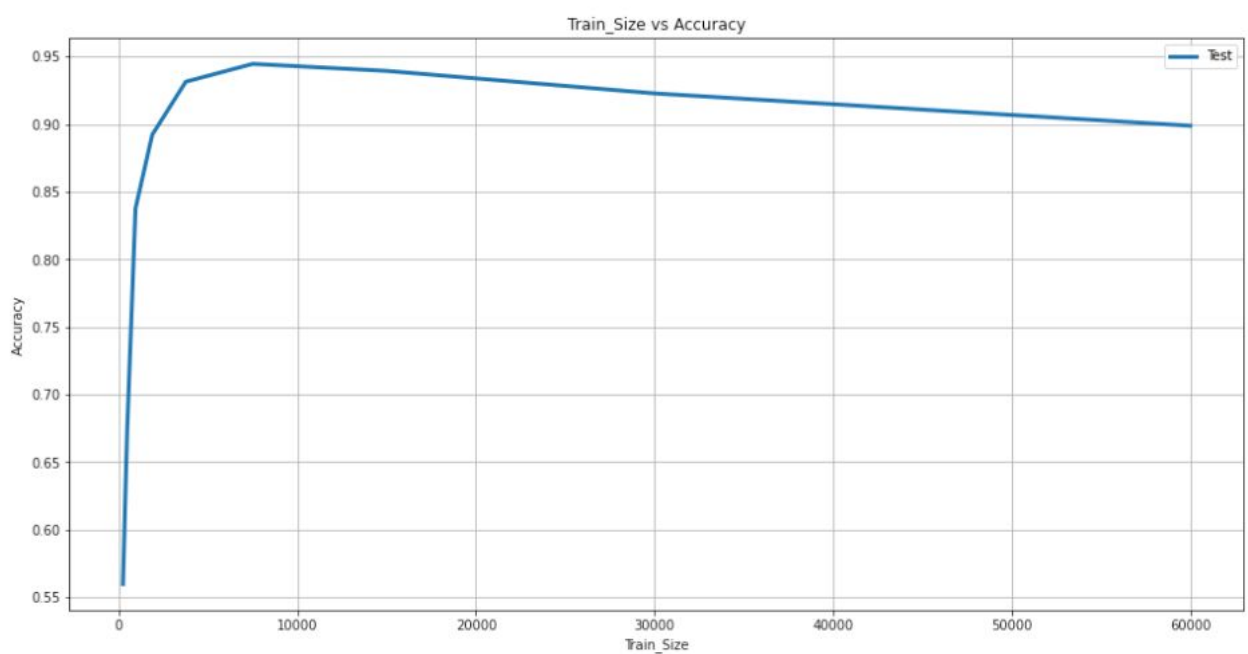
```
Train time elapsed: 14.82s
Test time elapsed: 0.00s
Training accuracy: 89.89%
                  === Classification Report ===
              precision    recall  f1-score   support

           0       0.94      0.91      0.93      5923
           1       0.86      0.98      0.92      6742
           2       0.93      0.87      0.90      5958
           3       0.87      0.86      0.87      6131
           4       0.91      0.90      0.91      5842
           5       0.88      0.84      0.86      5421
           6       0.93      0.94      0.93      5918
           7       0.95      0.91      0.93      6265
           8       0.88      0.85      0.87      5851
           9       0.84      0.91      0.88      5949

    accuracy                           0.90     60000
   macro avg       0.90      0.90      0.90     60000
weighted avg       0.90      0.90      0.90     60000

Accuracy of Classifier on Test Images:  0.8989
```



	Accuracy
Digit 0	91.018065
Digit 1	98.264610
Digit 2	86.690164
Digit 3	85.956614
Digit 4	90.243067
Digit 5	84.117322
Digit 6	94.035147
Digit 7	90.630487
Digit 8	85.199111
Digit 9	91.158178

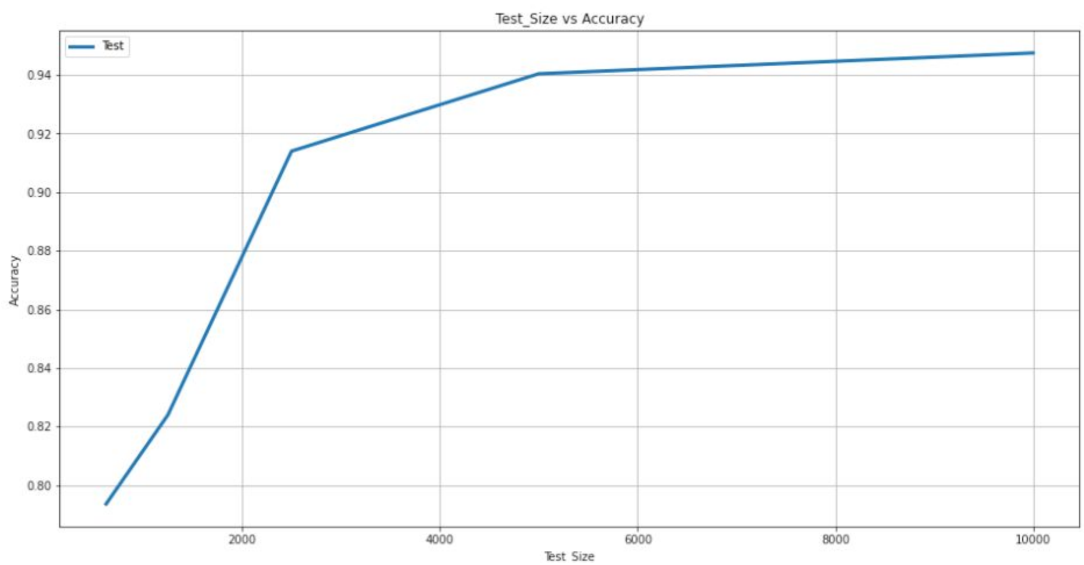
b) CategoricalNB Changing training size with constant test size

Train Size	Accuracy	Time	Test Size
60000	0.8989	18.06s	10000
30000	0.9223	9.51s	10000
15000	0.9394	4.09s	10000
7500	0.9446	1.91s	10000
3750	0.9314	1.06s	10000
1875	0.8922	0.75s	10000
937	0.8377	0.65s	10000
468	0.6730	0.56s	10000
234	0.5595	0.56s	10000
117	0.5348	0.42s	10000

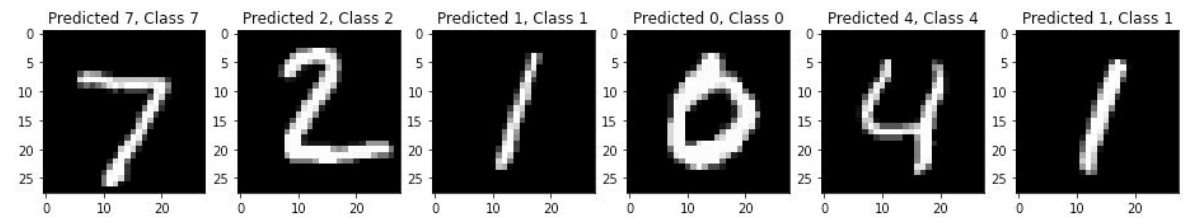


c) Changing testing size with constant train size.

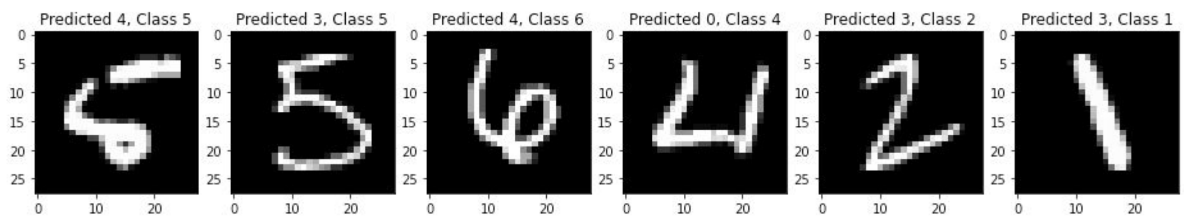
Test Size	Accuracy	Time	Train Size
10000	0.8413	15.63s	60000
5000	0.7998	13.26s	60000
2500	0.7956	12.86s	60000
1250	0.8024	13.08s	60000
675	0.8096	12.83s	60000
337	0.8046	11.83s	60000



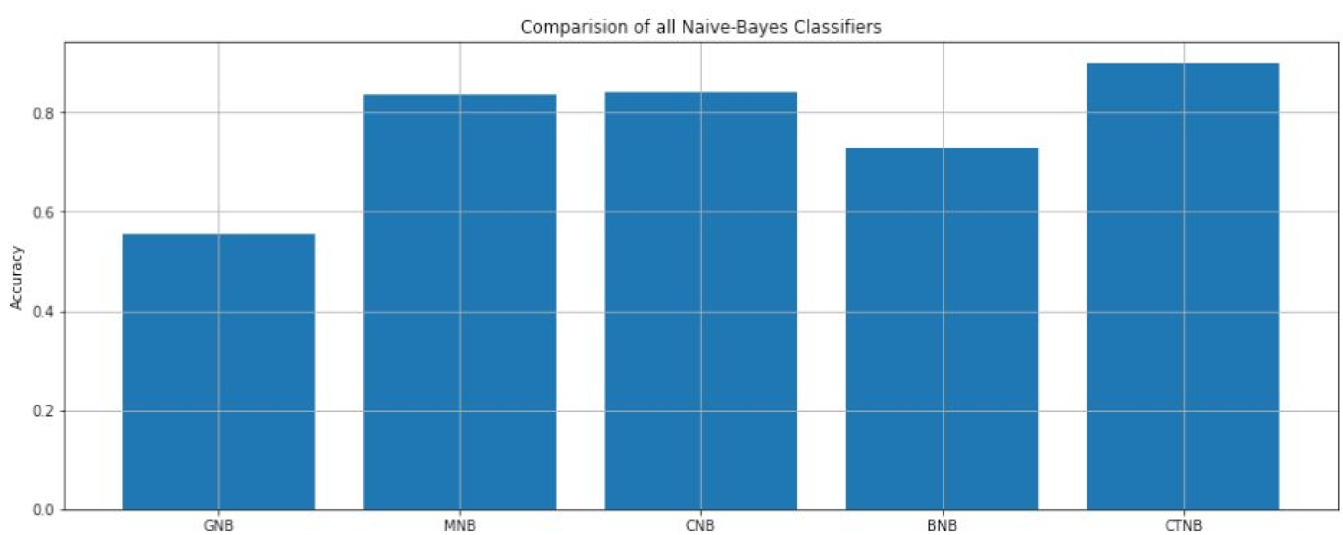
Correct Prediction:



Wrong Prediction:



Comparison of the all Naive-Bayes Classifier Results



Conclusion and Discussion:

I implemented a Naive Bayes classifier from scratch and applied it on the MNIST dataset.

The overall error is minimum of 11.02% which is about 89.98% accuracy with categorical Naive Bayes classifier. The digits 0 and 1 tend to have minimal error rate compared to the other digits, this can be attributed to the fact that the digits 0 and 1 have low variabilities. Commonly, people tend to write digits 0 and 1 in the right way. Digits 4 & 5 have high error rates, visualizing some of them show that they have high variations. Digit 4 is mostly misclassified as 9, some people tend to write 4 in a way that it looks like 9. Also, digit 2 is mostly misclassified as 8. Digit 9 is mostly misclassified as either 4 or 8, likewise, digits 4 and 8 are also occasionally misclassified as 9. The negative in the Naive Bayes classifier is that it assumes that all the dimensions shown in the dataset are independent of one another. However, it’s not accurate. Generally, Naive Bayes did poorly on the MNIST dataset, this could be attributed to the independent assumption which is likely not to be correct. Query time is faster compared to KNN, however, KNN provided a better performance on the MNIST dataset. Naive Bayes doesn't perform well when there are repeated attributes or when attributes are not equally important, which is the case in the MNIST dataset.

Decision Tree Classifier

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. For instance, in the example below, decision trees learn from data to approximate a mnist dataset with a set of decision rules.

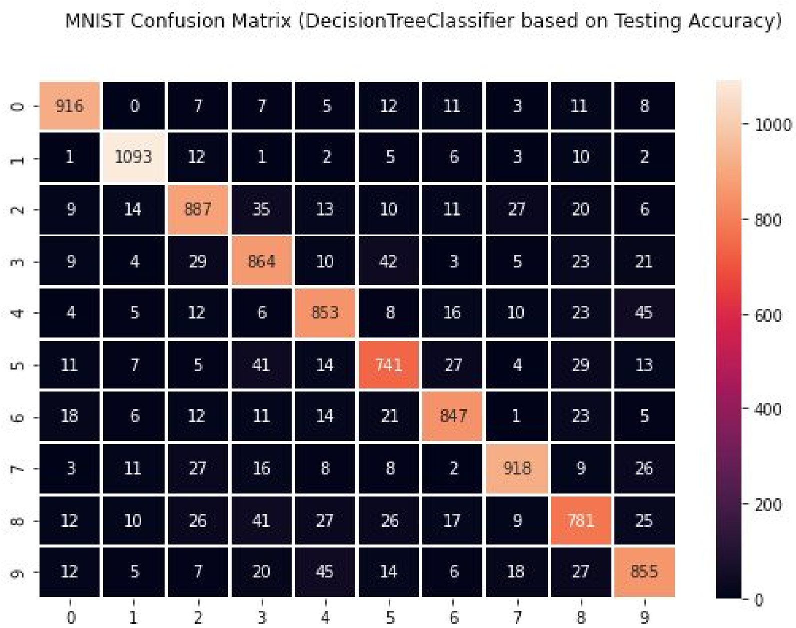
Experiments:

a) Decision tree performance with original dataset size

Figure[5]

```
Train time elapsed: 0.45s
Test time elapsed: 0.08s
Training accuracy: 100.00%
Testing accuracy: 87.55%
                  === Classification Report ===
              precision    recall  f1-score   support

           0       0.92      0.93      0.93       980
           1       0.95      0.96      0.95      1135
           2       0.87      0.86      0.86      1032
           3       0.83      0.86      0.84      1010
           4       0.86      0.87      0.86       982
           5       0.84      0.83      0.83       892
           6       0.90      0.88      0.89       958
           7       0.92      0.89      0.91      1028
           8       0.82      0.80      0.81       974
           9       0.85      0.85      0.85      1009

    accuracy                           0.88     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.88      0.88      0.88     10000

Accuracy of Classifier on Test Images:  0.8755
```

Figure[6]



	Accuracy
Digit 0	93.469388
Digit 1	96.299559
Digit 2	85.949612
Digit 3	85.544554
Digit 4	86.863544
Digit 5	83.071749
Digit 6	88.413361
Digit 7	89.299611
Digit 8	80.184805
Digit 9	84.737364

b) Accuracy and Depth Correlation

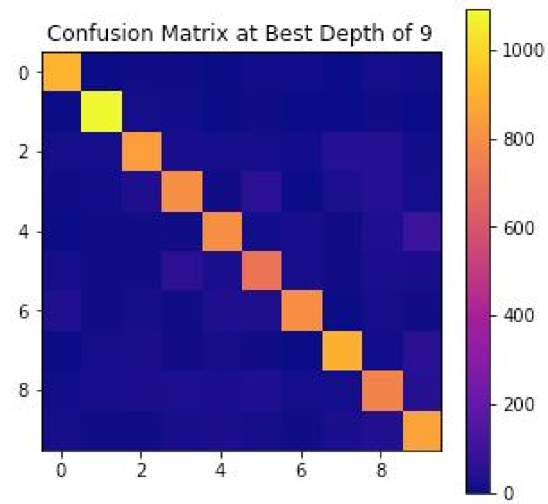
Depth	Accuracy	Time	Train & Test Size
1	0.1994	2.08s	60000 & 10000
2	0.3447	3.51s	60000 & 10000
3	0.4953	4.63s	60000 & 10000
4	0.5957	6.21s	60000 & 10000
5	0.6747	7.63s	60000 & 10000
6	0.7416	9.09s	60000 & 10000
7	0.7853	10.20s	60000 & 10000
8	0.8185	11.37s	60000 & 10000
9	0.8253	12.16s	60000 & 10000
10	0.885	13.34s	60000 & 10000

Figure[7]

Figure[8]

Classification Details of Best Depth(9)

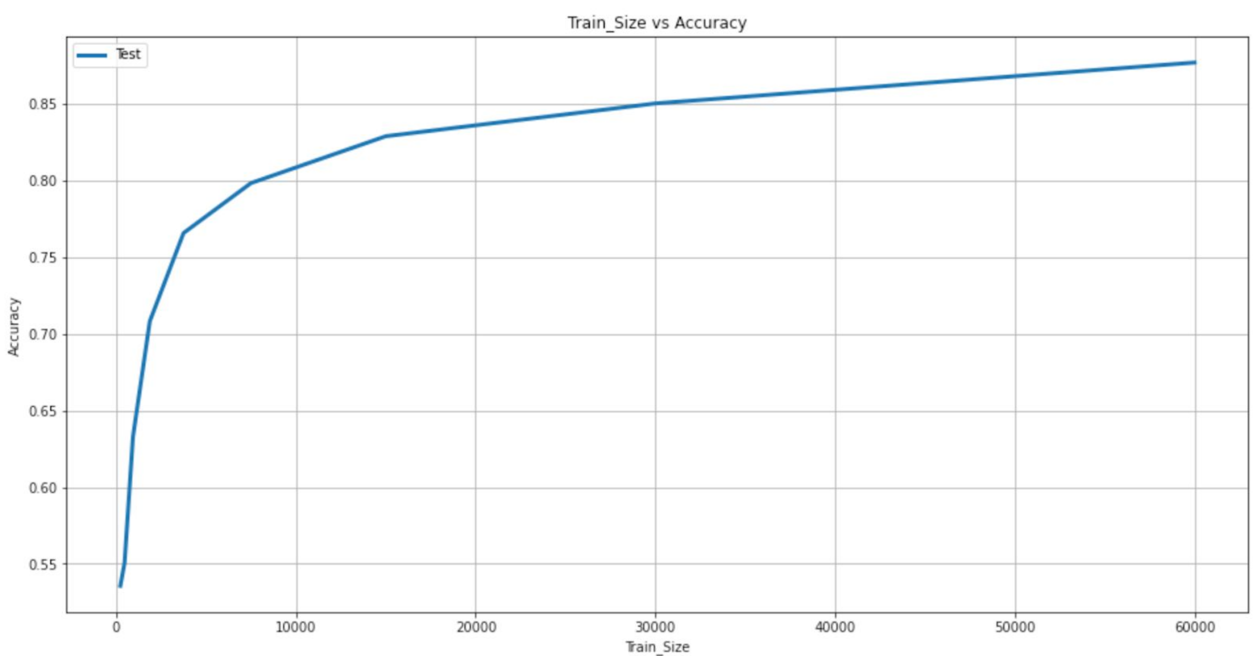
	precision	recall	f1-score	support
0	0.90	0.93	0.92	980
1	0.93	0.96	0.95	1135
2	0.87	0.82	0.84	1032
3	0.83	0.80	0.81	1010
4	0.85	0.82	0.83	982
5	0.80	0.80	0.80	892
6	0.90	0.84	0.87	958
7	0.88	0.88	0.88	1028
8	0.77	0.79	0.78	974
9	0.77	0.85	0.81	1009
accuracy			0.85	10000
macro avg	0.85	0.85	0.85	10000
weighted avg	0.85	0.85	0.85	10000



c) Changing training size with constant test size

Train Size	Accuracy	Time	Test Size
60000	0.8772	32.95s	10000
30000	0.8505	12.02s	10000
15000	0.8291	5.15s	10000
7500	0.7985	1.98s	10000
3750	0.7659	0.77s	10000
1875	0.7084	0.35s	10000
937	0.6329	0.20s	10000
468	0.5505	0.16s	10000
234	0.5356	0.11s	10000
117	0.5232	0.08s	10000

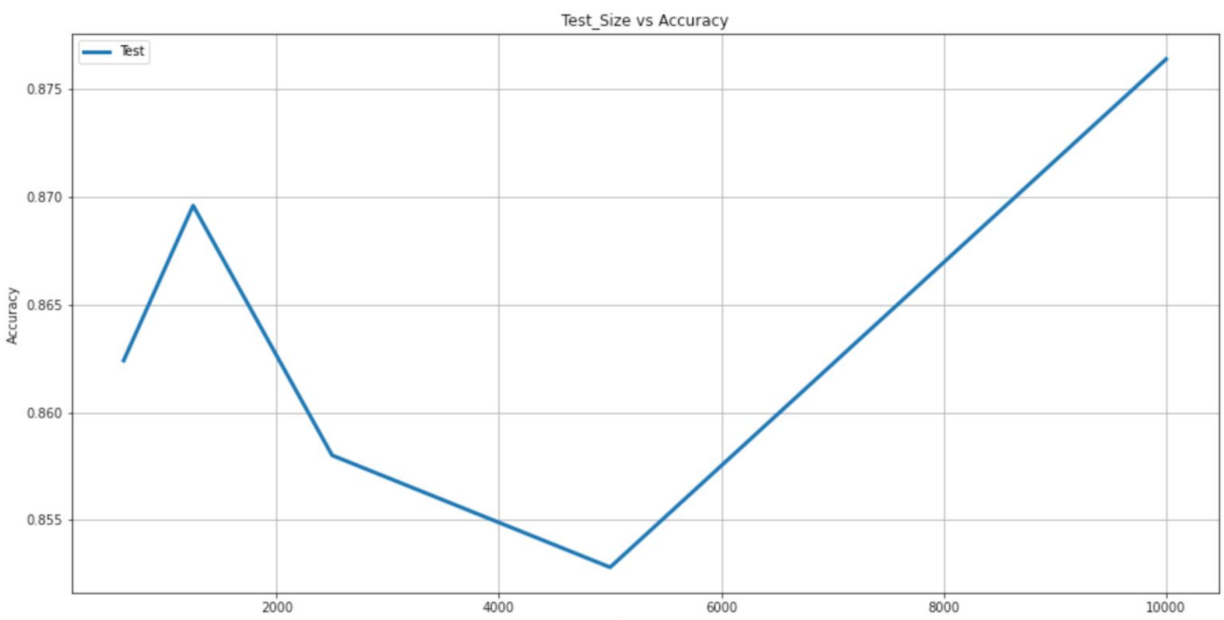
Figure[9]



d) Changing testing size with constant train size.

Test Size	Accuracy	Time	Train Size
10000	0.8764	34.05s	60000
5000	0.8528	32.46s	60000
2500	0.858	32.86s	60000
1250	0.8696	34.00s	60000
675	0.8624	33.11s	60000
337	0.8746	31.83s	60000

Figure[10]



Result and Discussion

I executed the decision tree classification model on all of the digits (0-9). Figure5 shows the test results after training the machine on this dataset. The overall recognition rate of the test database is 87.55%. Figure6 displays that some of the digits are not recognized by the machine learning model. For example, 8 were predicted as 5 and the reason behind this is the similarity of handwriting style between 5 and 8. Besides, The algorithm is executed with different depth values between 1 and 10. The graphical representation of the accuracy of classification in different dept values are shown in figure 7 and the overall classification results are listed out in the table. From the figure7 table, it is clearly evident that the optimal depth of value is 10 with %88.55 accuracy rate. Moreover, if we increase the depth value, we may get better results. Figure 8 shows linear growing on accuracy based on depth value. Furthermore, the decision tree algorithm is executed with different train size and test size parameters. Figure[9] and Figure[10] concludes how to train size and test size affects the overall accuracy result.

System Information:

Processor: 2,5 GHz Dual Core Intel Core i5

Memory: 8 GB 1600 MHz DDR3

GPU: Intel HD Graphics 4000 1536 MB

OS: MacOS Catalina 10.15.4

References

K-Nearest Neighbors Classifier

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Naive Bayes Classifier

https://scikit-learn.org/stable/modules/naive_bayes.html

Decision Tree Classifier

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

MNIST Dataset

http://yann.lecun. com/exdb/mnist/

Scikit-learn: Machine Learning Python:Journal of Machine Learning Research (2011).

http://www.jmlr.org/papers/v12/pedregosa11a.html

[11] MNB Definition

http://blog.datumbox.com/machine-learning-tutorial-the-naive-bayes-text-classifier/

[12] BNB Definition

http://blog.datumbox.com/machine-learning-tutorial-the-naive-bayes-text-classifier/